Docket: 50277-2433
OID 2003-333-01

Patent

UNITED STATES PATENT APPLICATION

FOR

COST-BASED OPTIMIZER FOR AN XML DATA REPOSITORY WITHIN A DATABASE

INVENTORS:

FEI GE
SIVASANKARAN CHANDRASEKAR
NIPUN AGARWAL
RAVI MURTHY
ERIC SEDLAR

ASSIGNEE:

ORACLE INTERNATIONAL CORPORATION
500 ORACLE PARKWAY
REDWOOD SHORES, CA 94065

PREPARED BY:

HICKMAN PALERMO TRUONG & BECKER LLP
1600 WILLOW STREET
SAN JOSE, CALIFORNIA 95125
(408) 414-1080

"Express Mail" mailing label number EV322193005US

Date of Deposit April 21, 2004

COST-BASED OPTIMIZER FOR AN XML DATA REPOSITORY WITHIN A DATABASE

FIELD OF THE INVENTION

[0001] The present invention relates generally to database systems and, more specifically, to a cost-based optimizer for an XML data repository within a relational database management system (RDBMS).

BACKGROUND OF THE INVENTION

DATABASE QUERY OPTIMIZER

[0002] In a database system, data is stored in one or more data containers, each container contains records, and the data within each record is organized into one or more fields. In relational database systems, the data containers are referred to as tables, the records are referred to as rows, and the fields are referred to as columns. In object oriented databases, the data containers are referred to as object classes, the records are referred to as objects, and the fields are referred to as attributes. Other database architectures may use other terminology.

[0003] A database management system (DBMS) retrieves and manipulates data in response to receiving a database statement. Typically, the database statement conforms to a database language, such as the Structured Query Language (SQL). A database statement can specify a query operation, a data manipulation operation, or a combination thereof. A database statement that specifies a query operation is referred to herein as a query.

[0004] When a DBMS receives a query, the DBMS may generate an execution plan. An execution plan is important because it defines the steps and operations performed by a DBMS to service a request. DBMSs often include an optimizer for generating execution

plans that are optimized for efficiency. When determining what steps to include in an execution plan, and the order in which the steps are performed, a DBMS accounts for many factors that affect efficiency. One important factor that is considered is the computational cost associated with executing a query according to a given execution plan. A cost-based optimizer (CBO) evaluates all possible data access paths for a query and determines the most efficient execution plan based on the cost of all access paths.

[0005] For example, a query with two ANDed predicates will request rows that satisfy both predicates. If the column(s) in the first predicate is indexed, then a DBMS may generate an execution plan that uses the index to access data more efficiently.

[0006] To determine an efficient execution plan for a query, the query optimizer relies on persistently stored statistics to estimate the costs of alternative execution plans, and chooses the plan with the lowest overall estimated cost. The statistics are computed and stored before the query is received. Statistics are used to estimate important optimizer cost parameters such as the selectivity of various predicates and predicate clauses (e.g., the fraction or percentage of rows in a table that match some condition represented in a predicate or predicate clause). Examples of statistics include table cardinalities (the number of rows in a table), the number of distinct values for a column, the minimum and maximum values in the column, and histograms, which is data that specifies the distribution of values in the columns, e.g., the number of rows that have particular column values for a column or the number of rows that have a column value that falls within a range. However, for some database statements, statistics needed by the query optimizer may not be available, such as statistics for certain repositories managed by the DBMS.

XML DATABASE

[0007] With support for XML type data as a native data type in information management systems, such as a relational database system (RDBMS) or object-relational database system (ORDBMS), the contents of XML documents can be stored in such systems. For example, in the context of a relational database, XML data may be stored in columns of a relational table and users can query the XML data via a SQL query.

[0008] One known implementation of an XML data repository, which provides the mechanisms for the storage of XML data in a RDBMS and access thereto, is referred to herein as an XML database ("XDB"). The key XDB-enabling technologies can be grouped into two major classes: (1) XML data type, which provides a native XML storage and retrieval capability strongly integrated with SQL; and (2) XDB repository, which provides foldering, access control, versioning, and the like, for XML resources.

[0009] The XML data type can be used as a datatype of a column of a relational table, and includes a number of useful methods to operate on XML data. XML type data can be stored, for example, as a LOB (large object) or according to object-relational storage. If stored as a LOB, XML data may be accessed via a text index, and if stored object relationally, XML data may be accessed via a btree index, for example. Some benefits that result from the XML data type include support for XML schemas, XPath searches, XML indexes, XML operators, XSL transformations, and XDB repository views (e.g., RESOURCE_VIEW and PATH_VIEW, described hereafter).

[0010] The XDB repository provides a repository for managing XML data. The XDB repository provides important functionality with respect to the XML data, for example, access control lists (ACL), foldering, WebDAV (Web-based Distributed Authoring and

Versioning), FTP (File Transfer Protocol) and JNDI (Java Naming and Directory Interface) access, SQL repository search, hierarchical indexing, and the like.

[0011] XDB repository views provide a mechanism for SQL access to data that is stored in the XDB. Data stored in XDB repository via protocols like FTP, WEBDAV or JNDI can be accessed in SQL via these views. XDB provides two repository views to enable SQL access to the repository: RESOURCE_VIEW and PATH_VIEW. Both views contain the resource properties, the path names and resource IDs. The PATH_VIEW has an additional column for the link properties.

[0012] With prior approaches to cost-based optimizers, the optimizers were unable to retrieve the real cost of a query on XDB repository views, so the optimizer relied on default statistics to choose a query execution plan. Since the CBO is not aware of the implementation of XDB repository views and user defined operators associated to the views, CBO can only estimate the default statistics, which is far from being accurate. Thus, the result is sub optimal query execution plans. For example, in the absence of an optimizer mechanism for an XDB repository, the CBO may choose a sub optimal query plan involving both a hierarchical index scan and a btree index scan, where the selectivity of the predicate with the XDB operator is very high while the selectivity of the predicate with the btree index on it is very low. In such a scenario, the optimal query plan would be a btree index scan followed by functional evaluation of the repository view operators.

[0013] The approaches described in this section are approaches that could be pursued, but not necessarily approaches that have been previously conceived or pursued. Therefore, unless otherwise indicated, it should not be assumed that any of the approaches described in this section qualify as prior art merely by virtue of their inclusion in this section.

BRIEF DESCRIPTION OF THE DRAWINGS

[0014] The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

[0015] FIG. 1 is a diagram that illustrates hierarchically-organized data;

[0016] FIG. 2 is a flow diagram that illustrates a method for computing computational costs associated with accessing XML resources stored in an XML database repository, according to an embodiment of the invention;

[0017] FIG. 3 is a block diagram that illustrates an example of an operating environment in which an embodiment of the invention may be implemented; and

[0018] FIG. 4 is a block diagram that illustrates a computer system upon which an embodiment of the invention may be implemented.

DETAILED DESCRIPTION OF EMBODIMENTS OF THE INVENTION

[0019] Techniques are described for applying cost-based optimizer functionality to an XML data (also referred to as XML resource) repository within a database.

[0020] In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, that the present invention may be practiced without these specific details. In other instances, well-known structures and devices are shown in block diagram form in order to avoid unnecessarily obscuring the present invention.

OVERVIEW OF EMBODIMENTS

[0021] Cost-based optimizer functionality for an XML database repository provides means for optimizing the execution of database queries that access XML resources in the database repository. Statistics about XML resources that are stored in the database repository are gathered, stored and utilized by a query optimizer to compute computational costs associated with each of multiple methods of accessing (referred to as "access paths") particular XML resources requested in a database query. Hence, the optimizer is able to select the most efficient query execution plan based on the costs of possible access paths. For example, with respect to one existing commercially available XML repository within a database management system, execution of SQL queries on XML database repository views (e.g., a RESOURCE_VIEW and/or PATH_VIEW) that contain operators such as UNDER_PATH and/or EQUALS_PATH can be optimized based on the cost of the possible access paths.

[0022] In one embodiment, specific statistics about the hierarchical structure of XML resources stored in the XML database repository are gathered, stored in a relational table in

the database management system, and used to compute the selectivity of query predicates and the index cost associated with traversing one or more indexes to access the requested XML resources. In a related embodiment, the function cost associated with the functional evaluation of the repository view operators is also computed. In one embodiment, the index cost comprises the computational cost associated with CPUs used for accessing the resources and the computational cost associated with reading data (or "disk") blocks in which portions of the index are stored.

HIERARCHICALLY-STRUCTURED XML DATA

[0023] XML data can be organized in a hierarchical structure which begins at a root node and in which each level of the structure contains one or more nodes. Each node is either a container node (or simply a "container") or a file. For example, a container node may represent a folder or directory, whereas a file has contents. An XML file contains XML elements and attributes. Files and empty containers are all leaf nodes.

[0024] FIG. 1 is a diagram that illustrates hierarchically-organized data. In FIG. 1, root 102 is the root node; container 104, container 106, container 110, and container 116 are container nodes; and file 108, file 112, file 114 and empty container 104 are leaf nodes. Container 104 is illustrated as an empty container node, e.g., a folder containing no files, and is, therefore, considered a leaf node. FIG. 1 is a simplified representation of a set of hierarchically-structured data, because the structure of XML data that is stored in an XML repository (see XDB 304 of FIG. 3) within a database management system (see DBMS 302 of FIG. 3) is typically much more complex.

[0025] Since XML files, referred to herein as "resources" or "data", can be organized in a hierarchical structure, a hierarchical index may be constructed for the XML files. As

embodiments are described hereafter, at least a portion of the cost of accessing a given resource stored in an XML database repository includes the cost of accessing the given resource from a table in which the given resource is stored, by traversing an index on the table. One technique for indexing hierarchically-structured resources is to construct and maintain a hierarchical index as described in U.S. Patent No. 6,427,123 entitled "Hierarchical indexing for accessing hierarchically organized information in a relational system" and U.S. Patent No. 6,571,231 entitled "Maintenance of hierarchical index in relational system."

METHOD FOR COMPUTING COSTS OF DATA ACCESS PATHS

[0026] FIG. 2 is a flow diagram that illustrates a method for computing computational costs associated with accessing XML resources stored in an XML database repository, according to an embodiment of the invention. As previously discussed, one known implementation of mechanisms for the storage of XML data in a RDBMS, and access thereto, is referred to herein as an XML database ("XDB"), part of which is an XDB repository. The term "XDB repository" is used to reference a particular implementation of a repository for XML data within a relational database management system, and the term "XML database repository" is used herein to generally reference any implementation of a repository for XML data within a database management system. The method illustrated in FIG. 2 is in the context of an XML database repository and, therefore, applicable to any implementation of a repository for XML data within a database management system.

GATHERING STATISTICS ABOUT XML RESOURCES

[0027] At block 202, statistics about XML resources that are stored in an XML database repository are gathered. When a node is said to be "under" a particular node, it means (1) that the node is at a level, in a hierarchy in which the XML data is organized, further from

the root node than the particular node, and (2) that the node is accessible from the database repository via a path through the particular node. In reference to FIG. 1, file 114 and file 116 are under container 110; container 110 and file 112 are under container 106; and so on.

[0028] In one embodiment, the following statistics are gathered, to provide bases for computing computational costs associated with one or more access paths to a given XML resource stored in an XML database repository. Furthermore, examples of column names for a table or tables in which each of the respective statistics are stored, are provided in brackets for subsequent reference thereto. Such statistics support cost computations for accessing a node, e.g., a file or document, in which a requested XML resource is logically stored (e.g., selectivity, as described hereafter). Other mechanisms are utilized to compute costs for accessing a particular resource associated with a node (e.g., index cost, as described hereafter).

[0029] (1) Total number of nodes (e.g., files and containers) under a particular node [resoid], i.e., at all levels under the particular node; [total_rows].

[0030] (2) Immediate number of nodes (e.g., files and containers) under a particular node [resoid], i.e., at the level immediately under the particular node; [fan_out].

[0031] (3) Total number of containers under a particular node [resoid], i.e., at all levels under the particular node; [total_containers].

[0032] (4) Immediate number of containers under a particular node [resoid], i.e., at the level immediately under the particular node; [immediate_containers].

[0033] (5) The depth of a particular node, i.e., how many levels from the root node; [depth].

[0034] With the known implementation of the XDB repository, statistics about XML resources stored in the XDB repository can be collected or deleted via an ANALYZE

command, described in Chapter 30 of "Oracle9*i* Database Administrator's Guide Release 1 (9.0.1), Part Number A90117-01", or a DBMS_STATS package, described in Chapter 3 of "Oracle9*i* Database Performance Guide and Reference Release 1 (9.0.1), Part Number A87503-02". Other implementations can collect statistics using similar functionalities.

[0035] In one embodiment, one or more of the foregoing statistics are gathered for each container node, rather than all of the statistics. Hence, a particular implementation of the techniques described herein may not gather or use all of the foregoing five statistics for cost computations.

[0036] In one embodiment, only queries that contain predicates or operators with a depth of one or infinity are optimized, because such queries represent the majority of practical scenarios. A depth of infinity is the default depth of a predicate or operator when a depth is not specified.

STORING STATISTICS ABOUT XML RESOURCES

[0037] At block 204, the statistics are stored. In one embodiment, the statistics are stored in a relational table of a database of which the XML database repository is part, such as a relational table in a RDBMS. Hence, the resoid column of such a table links the statistics row to a node in the repository. Based on the embodiment in which optimization is only performed for queries with a depth of one or infinity, total_rows and total_containers store statistics relevant to infinite depth, and fan_out and immediate_containers store statistics relevant to a depth of one. The statistics stored in the depth column are used to enhance search and identification of a given resource.

[0038] In one embodiment, the table in which the statistics are stored (the "statistics table") is implemented as a schema-based XML table. Each row in the statistics table

corresponds to a node in the XML data hierarchy. This approach leverages the potential of keeping the statistics table in a generally consistent form with the rest of the data hierarchy stored in the XML database repository, which is XML-enabled.

[0039] Options for implementing storage of the statistics include (1) storing the statistics in a hierarchical index on the table(s) in which the resources are stored ("resource table"), such as a hierarchical index table as described in U.S. Patent No. 6,427,123; (2) storing the statistics in the resource table; and (3) storing the statistics in a table separate from the resource table. In an embodiment, the statistics are stored in a separate table (option 3 above), such as a schema-based table of XML data type.

COMPUTING COSTS OF XML RESOURCE ACCESS PATHS

[0040] At block 206, in response to a request for access to XML resources from the database repository, a computational cost associated with one or more methods for accessing the requested XML resources (i.e., access paths) is computed, based on at least a portion of the statistics that are gathered at block 202. For example, the computational cost of a SQL query is computed as described hereafter, in response to receiving the SQL query at a database server.

[0041] In one embodiment, the computational cost comprises selectivity and index cost (both are described hereafter). In an alternative embodiment, the computational cost comprises one from the group consisting of selectivity and index cost. However, the specific parameters that are calculated as components of the computational cost may vary from implementation to implementation.

SELECTIVITY

[0042] In one embodiment, an optimizer uses the statistics stored in the statistics table to calculate the selectivity of predicates containing operators on the XML database repository. For example, the optimizer calculates the selectivity of predicates containing path-related operators associated with the hierarchy in which the XML data is organized.

[0043] In one embodiment, a path-related operator is an operator that determines whether a particular XML resource can be located in a specified column of the resource table in said database repository through a particular specified path through a portion of the hierarchy. In other words, the operator determines whether the particular XML resource is "under" the specified path. With the known implementation of the XDB repository, such an operator is referred to as UNDER_PATH; however, embodiments are not limited to use of that specific operators.

[0044] In one embodiment, a path-related operator is an operator that determines whether a particular XML resource can be located in a specified column of the resource table in said database repository at a terminal location of a particular specified path through a portion of the hierarchy. With the known implementation of the XDB repository, such an operator is referred to as EQUALS_PATH; however, embodiments are not limited to use of that specific operators.

[0045] Selectivity is defined as a percentage number between zero (0) and one hundred (100). Given a predicate in the following form or with a similar operator specified, where depth is not specified (default is infinity),

UNDER_PATH(res, '/p1') = 1,

where "res" is the column name and "/p1" is the specified path,

the selectivity of the predicate can be calculated as:

<selectivity> = (<total_rows ('/p1') > / <total_rows(root)>) * 100,

where the "root" is the node of the hierarchy from which all paths originate.

[0046] Given a predicate in the following form or with a similar operator specified, where depth is set to 1,

UNDER_PATH(res, 1, '/p1") = 1,

the selectivity can be calculated as:

<selectivity> = (<fan_out('/p1')> / <total_rows(root)>) * 100.

[0047] For a predicate that contains an operator like EQUALS_PATH() = 1, or with a similar operator specified, the number of rows in the output is always one (1) and, therefore, its selectivity can be calculated as:

<selectivity> = (1 / <total_rows(root)>) * 100.

[0048] For a predicate that contains an operator such as UNDER_PATH('/') or with a similar operator specified, without specified depth (default is infinity), the selectivity is always 100%, so there is no need to query the statistics table in this case.

INDEX COST

[0049] In an embodiment, the optimizer defines a cost function for hierarchical indexes on the XML database repository. Whenever an index scan on the hierarchical index (sometimes referred to as traversing the index) is part of a valid access path, the optimizer invokes this index cost function, which computes a composite cost comprising (1) a computational cost value associated with one or more CPUs that are used for traversing the index (e.g., a CPU cost); and/or (2) a computational cost value associated with reading data blocks in which portions of the index are stored (e.g., an I/O cost). Furthermore, a network cost may be included in the index cost.

CPU Cost

[0050] The CPU cost portion of the index cost is an estimate of the CPU cost of accessing an XML resource based on the elapsed time to access the resource. A CPU cost function approximates the number of CPU instructions corresponding to a specified time interval. The CPU cost function takes as input the elapsed time of the index scan process, measures CPU units by multiplying the elapsed time by the processor speed of the machine, and returns the approximate number of CPU instructions that should be associated with the index scan.

[0051] With a known commercial database management system, the CPU cost associated with a hierarchical index scan to access XML resources can be computed via an ESTIMATE_CPU_UNITS command, described in Chapter 31 of "Oracle9*i* Supplied PL/SQL Packages and Types Reference Release 1 (9.0.1), Part Number A89852-02". However, embodiments are not limited to use of that specific command, as other implementations can compute the CPU cost using similar functionalities.

I/O Cost

[0052] I/O cost is the number of data blocks associated with the hierarchical index table read by the predicate operators of interest, for example, the UNDER_PATH operator. In one embodiment, only the data blocks occupied by containers are of interest when computing the I/O cost. In most scenarios, there is one block per container. However, in scenarios in which a container is across multiple blocks, each block is counted as a different container in the total_containers column in the statistics table.

[0053] When queries with a specified depth exceeding 1 are optimized, statistics about the median depth of the various paths of the hierarchy under a given node (median_depth) and the maximum depth of the various paths of the hierarchy under a given node

(max_depth) are useful. The distribution of containers under a given node is not included in the statistics table, but the total number of data blocks occupied by containers of interest can be estimated from the following three values: total_containers, median_depth and max_depth.

[0054] Given a predicate in the following form or with a similar operator specified, where depth is not specified,

UNDER_PATH(res, '/p1') = 1,

the I/O cost can be calculated as follows:

<IO_cost> : = <total_containers('/p1')>

[0055] Given a predicate in the following form or with a similar operator specified,

UNDER_PATH(res, 1, '/p1') =1,

where depth is 1, the I/O cost can be calculated as follows:

<IO_cost> := <immediate_containers('/p1)>

[0056] For a predicate containing EQUALS_PATH or with a similar operator specified, the I/O cost is equal to the number of components in the path. For example, accessing a resource specified by "/sys/home/foo/bar" will access three containers ("sys", "home" and "foo") and, therefore, will access three data blocks, to reach the resource "bar".

FUNCTION COST

[0057] In one embodiment, another component of the computational cost of accessing an XML resource in an XML database repository is a function cost. The function cost is calculated by the optimizer when functional implementations of path-based operators, such as UNDER_PATH and EQUALS_PATH and similar functions, are valid access paths.

[0058] The cost of functional implementation of the UNDER_PATH or similar functioning path-based operator is often primarily from a CONNECT BY or similar functioning clause, which retrieves all the paths for a given resource. Therefore, in one embodiment, the I/O cost and CPU cost of the CONNECT BY are used to calculate the function cost. For example, the I/O cost and CPU cost for the CONNECT BY clause can be determined from an internal EXPLAIN PLAN on the CONNECT BY, i.e., the function cost is retrieved by taking advantage of existing functionality offered by the cost based optimizer. In another embodiment, the function cost is calculated similarly to the index cost described herein, i.e., based on the I/O cost and the CPU cost associated with the functional implementations of the path-based operators.

[0059] In one embodiment, however, the I/O cost and CPU cost retrieved from the plan table are multiplied by a constant factor to reflect the fact that the cost of a functional implementation also includes other overhead and is relative to the function cost of an index scan as well as other indexes.

OPERATING ENVIRONMENT EXAMPLE

[0060] FIG. 3 is a block diagram that illustrates an example of an operating environment in which an embodiment may be implemented. The techniques described herein can be implemented in a database management system 302, such as a relational database management system (RDBMS). The database management system 302 comprises a database server 304 and a database 306.

[0061] Generally, the database 306 comprises data and metadata that is stored on a persistent memory mechanism, such as a set of hard disks that are communicatively coupled to the database server 304. Such data and metadata may be stored in database 306 logically,

for example, according to relational database constructs, multidimensional database constructs, or a combination of relational and multidimensional database constructs. Database 306 comprises a XML database repository 312 for storing XML data, as described herein and on which XML data access requests are made.

[0062] Database server 304 is a combination of integrated software components and an allocation of computational resources (such as memory and processes) for executing the integrated software components on a processor, where the combination of the software and computational resources are used to manage a particular database, such as database 306. Among other functions of database management, a database server such as database server 304 typically governs and facilitates access to database 306 by processing requests from clients to access the data in database 306. Database server 304 can be implemented on one or more conventional computer systems, such as computer system 400 illustrated in FIG. 4.

[0063] Database server 304 comprises functional components for performing the techniques described herein. These functional components are referred to as a statistics module 308 and a cost-based optimizer (CBO) 310. Each of statistics module 308 and CBO 310 comprise one or more sequences of instructions which, when executed, cause one or more processors to perform certain actions. For example, statistics module 308 comprises instructions for performing blocks 202 and 204 of FIG. 2 and CBO 310 comprises instructions for performing block 206 of FIG. 2.

HARDWARE OVERVIEW

[0064] FIG. 4 is a block diagram that illustrates a computer system 400 upon which an embodiment of the invention may be implemented. Computer system 400 includes a bus 402 or other communication mechanism for communicating information, and a processor 404

coupled with bus 402 for processing information. Computer system 400 also includes a main memory 406, such as a random access memory (RAM) or other dynamic storage device, coupled to bus 402 for storing information and instructions to be executed by processor 404. Main memory 406 also may be used for storing temporary variables or other intermediate information during execution of instructions to be executed by processor 404. Computer system 400 further includes a read only memory (ROM) 408 or other static storage device coupled to bus 402 for storing static information and instructions for processor 404. A storage device 410, such as a magnetic disk, optical disk, or magneto-optical disk, is provided and coupled to bus 402 for storing information and instructions.

[0065] Computer system 400 may be coupled via bus 402 to a display 412, such as a cathode ray tube (CRT) or a liquid crystal display (LCD), for displaying information to a computer user. An input device 414, including alphanumeric and other keys, is coupled to bus 402 for communicating information and command selections to processor 404. Another type of user input device is cursor control 416, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 404 and for controlling cursor movement on display 412. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a plane.

[0066] The invention is related to the use of computer system 400 for implementing the techniques described herein. According to one embodiment of the invention, those techniques are performed by computer system 400 in response to processor 404 executing one or more sequences of one or more instructions contained in main memory 406. Such instructions may be read into main memory 406 from another computer-readable medium, such as storage device 410. Execution of the sequences of instructions contained in main

memory 406 causes processor 404 to perform the process steps described herein. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions to implement the invention. Thus, embodiments of the invention are not limited to any specific combination of hardware circuitry and software.

[0067] The term "computer-readable medium" as used herein refers to any medium that participates in providing instructions to processor 404 for execution. Such a medium may take many forms, including but not limited to, non-volatile media, volatile media, and transmission media. Non-volatile media includes, for example, optical, magnetic, or magneto-optical disks, such as storage device 410. Volatile media includes dynamic memory, such as main memory 406. Transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise bus 402. Transmission media can also take the form of acoustic or light waves, such as those generated during radio-wave and infra-red data communications.

[0068] Common forms of computer-readable media include, for example, a floppy disk, a flexible disk, hard disk, magnetic tape, or any other magnetic medium, a CD-ROM, any other optical medium, punchcards, papertape, any other physical medium with patterns of holes, a RAM, a PROM, and EPROM, a FLASH-EPROM, any other memory chip or cartridge, a carrier wave as described hereinafter, or any other medium from which a computer can read.

[0069] Various forms of computer readable media may be involved in carrying one or more sequences of one or more instructions to processor 404 for execution. For example, the instructions may initially be carried on a magnetic disk of a remote computer. The remote computer can load the instructions into its dynamic memory and send the instructions over a telephone line using a modem. A modem local to computer system 400 can receive the data on the telephone line and use an infra-red transmitter to convert the data to an infra-red

signal. An infra-red detector can receive the data carried in the infra-red signal and appropriate circuitry can place the data on bus 402. Bus 402 carries the data to main memory 406, from which processor 404 retrieves and executes the instructions. The instructions received by main memory 406 may optionally be stored on storage device 410 either before or after execution by processor 404.

[0070] Computer system 400 also includes a communication interface 418 coupled to bus 402. Communication interface 418 provides a two-way data communication coupling to a network link 420 that is connected to a local network 422. For example, communication interface 418 may be an integrated services digital network (ISDN) card or a modem to provide a data communication connection to a corresponding type of telephone line. As another example, communication interface 418 may be a local area network (LAN) card to provide a data communication connection to a compatible LAN. Wireless links may also be implemented. In any such implementation, communication interface 418 sends and receives electrical, electromagnetic or optical signals that carry digital data streams representing various types of information.

[0071] Network link 420 typically provides data communication through one or more networks to other data devices. For example, network link 420 may provide a connection through local network 422 to a host computer 424 or to data equipment operated by an Internet Service Provider (ISP) 426. ISP 426 in turn provides data communication services through the world wide packet data communication network now commonly referred to as the "Internet" 428. Local network 422 and Internet 428 both use electrical, electromagnetic or optical signals that carry digital data streams. The signals through the various networks and the signals on network link 420 and through communication interface 418, which carry

the digital data to and from computer system 400, are exemplary forms of carrier waves transporting the information.

[0072] Computer system 400 can send messages and receive data, including program code, through the network(s), network link 420 and communication interface 418. In the Internet example, a server 430 might transmit a requested code for an application program through Internet 428, ISP 426, local network 422 and communication interface 418.

[0073] The received code may be executed by processor 404 as it is received, and/or stored in storage device 410, or other non-volatile storage for later execution. In this manner, computer system 400 may obtain application code in the form of a carrier wave.

EXTENSIONS AND ALTERNATIVES

[0074] Alternative embodiments of the invention are described throughout the foregoing description, and in locations that best facilitate understanding the context of the embodiments. Furthermore, the invention has been described with reference to specific embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention. For example, embodiments were described herein in reference to SQL-based access to the XML database repository; however, the broad techniques described herein are applicable to other means for accessing XML resources within an XML database repository, such as with FTP (File Transfer Protocol) and HTTP (HyperText Transfer Protocol). Therefore, the specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

[0075] In addition, in this description certain process steps are set forth in a particular order, and alphabetic and alphanumeric labels may be used to identify certain steps. Unless

specifically stated in the description, embodiments of the invention are not necessarily limited to any particular order of carrying out such steps. In particular, the labels are used merely for convenient identification of steps, and are not intended to specify or require a particular order of carrying out such steps.